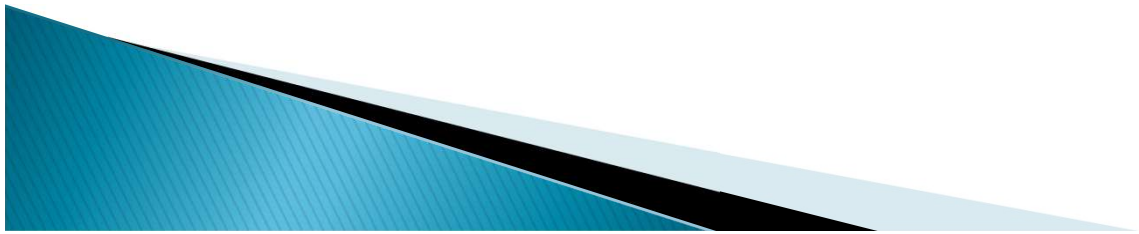
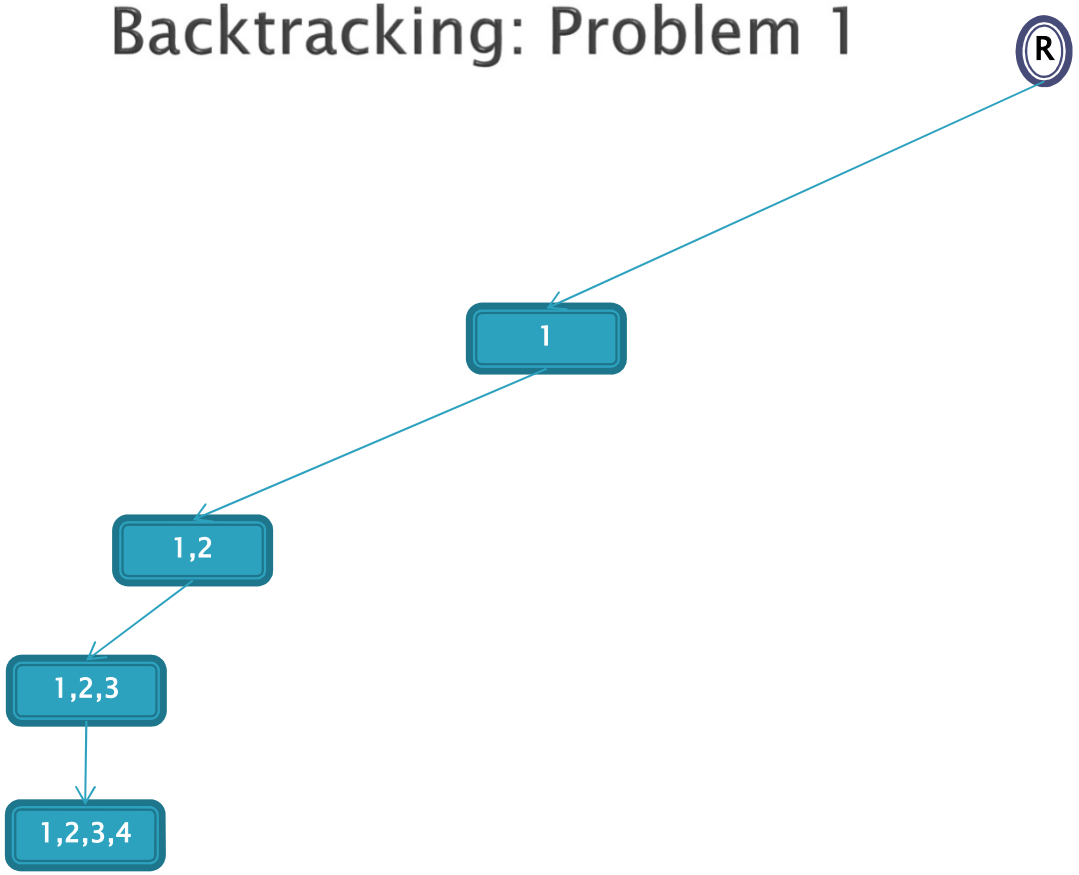
Backtracking: Problem 1

There are N different elements stored in a direct access structure (for example, a vector with the numbers 1, 2, 3, 4 and 5, or the string abcdefg) and we want to obtain all the different ways of placing those elements, it is say, you have to get all the permutations of the N elements. Design an algorithm that uses Backtracking to solve the problem.



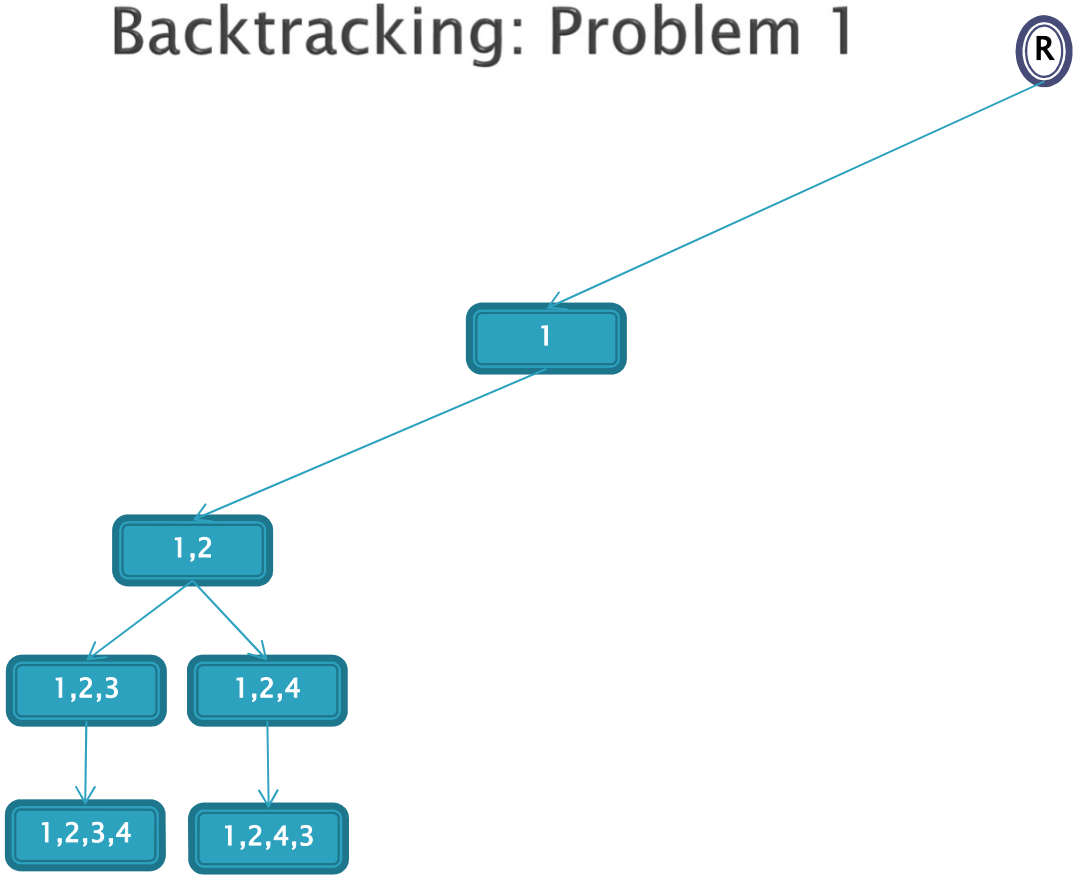
Backtracking: Problem 1



Input={1,2,3,4}

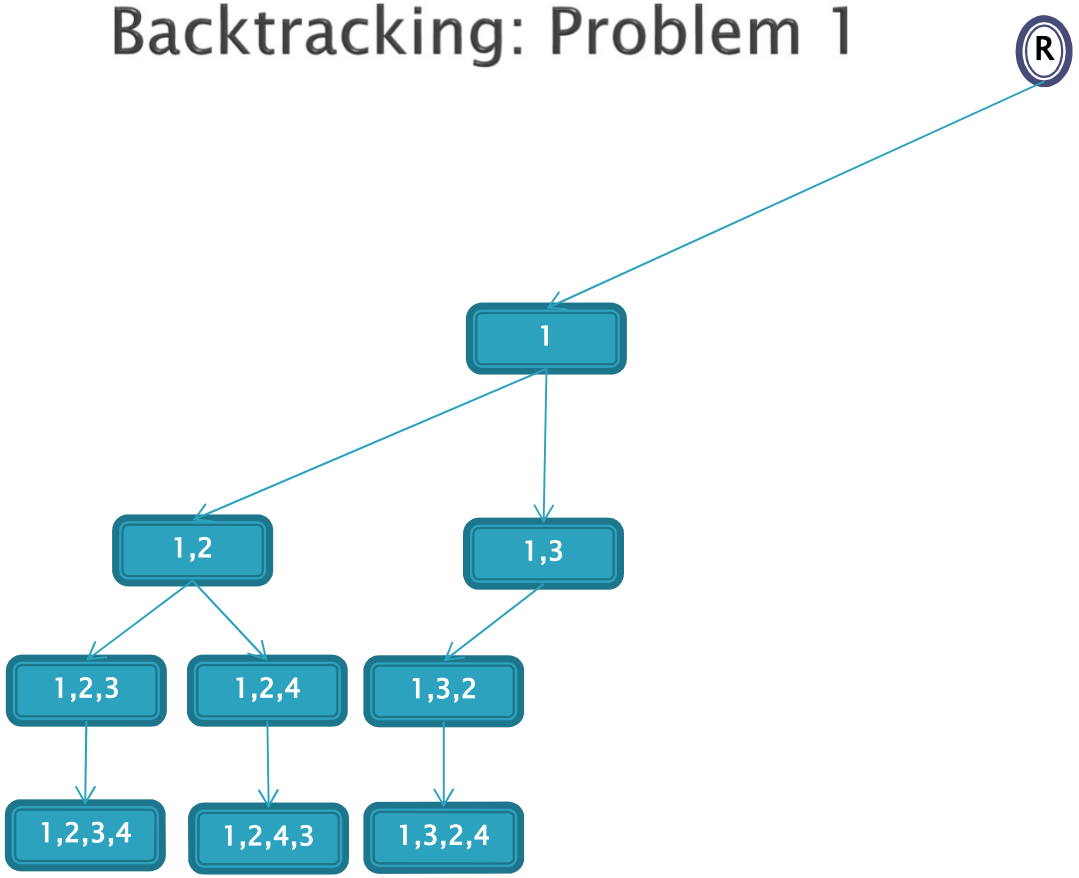
Backtracking: Problem 1

Input ={1,2,3,4}



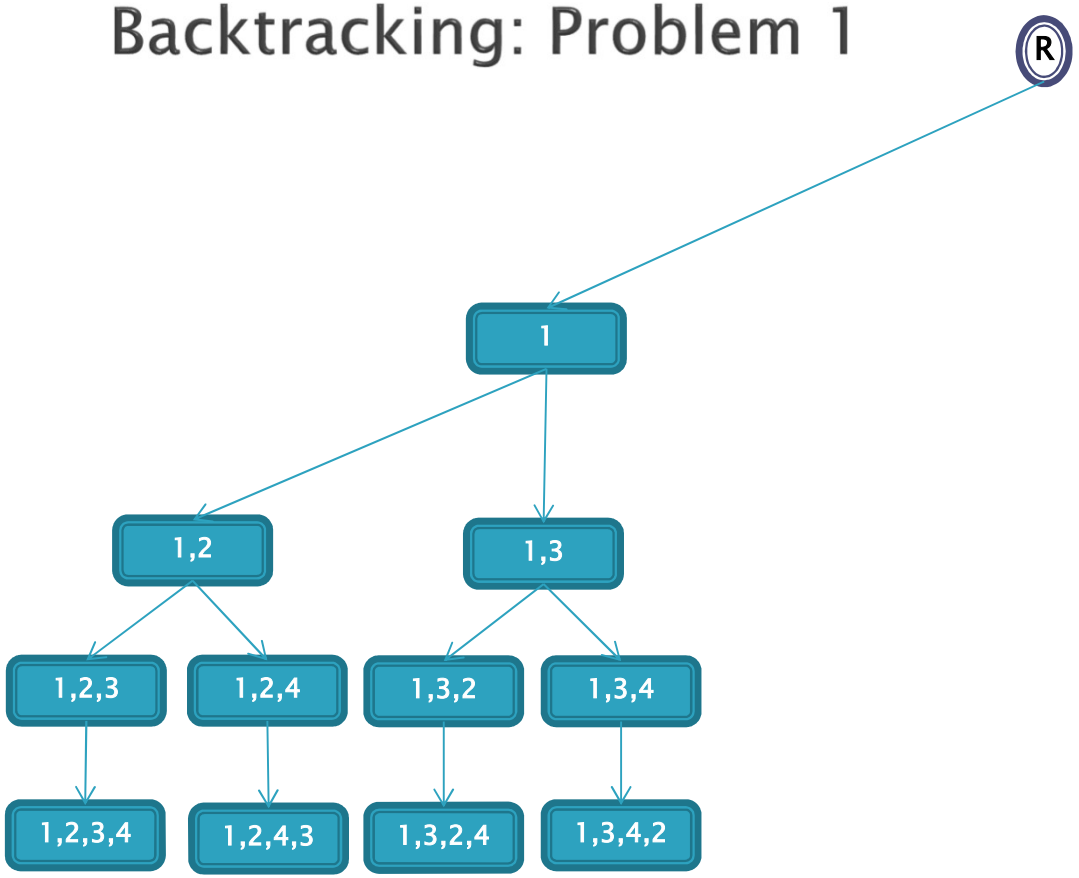
Backtracking: Problem 1

Input ={1,2,3,4}



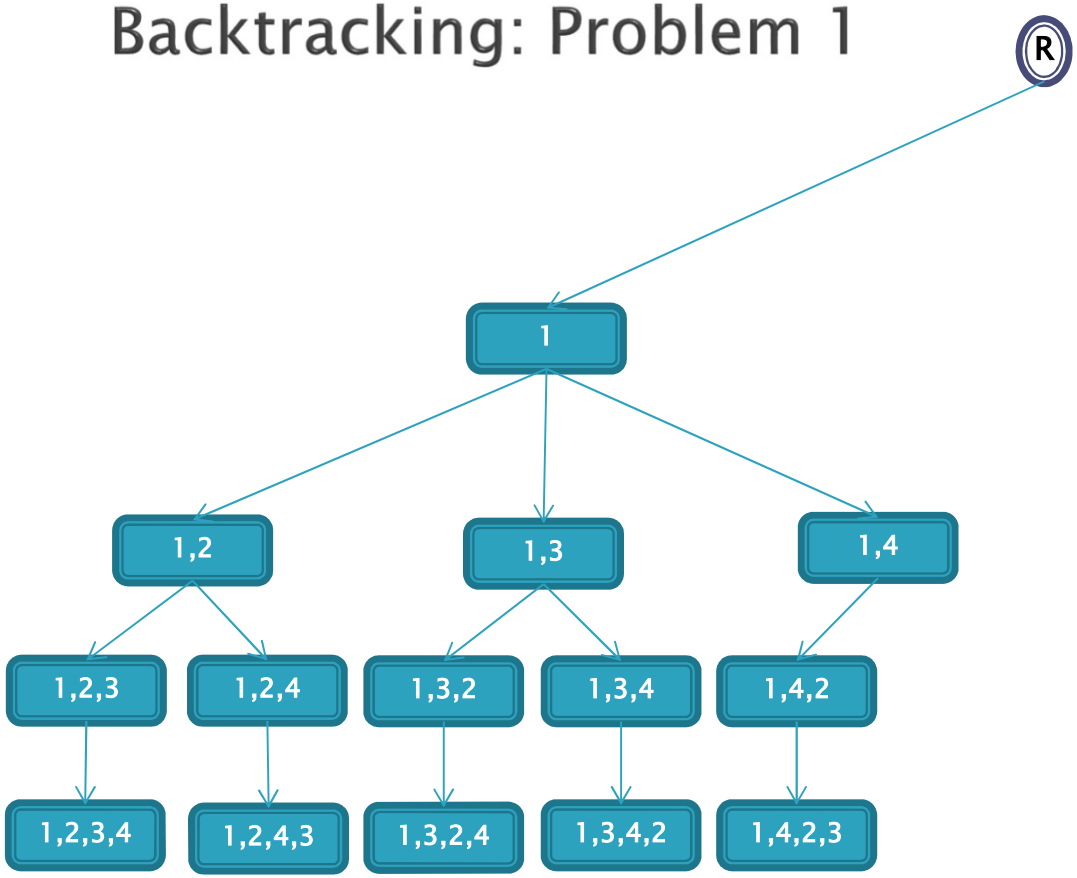
Backtracking: Problem 1

Input ={1,2,3,4}



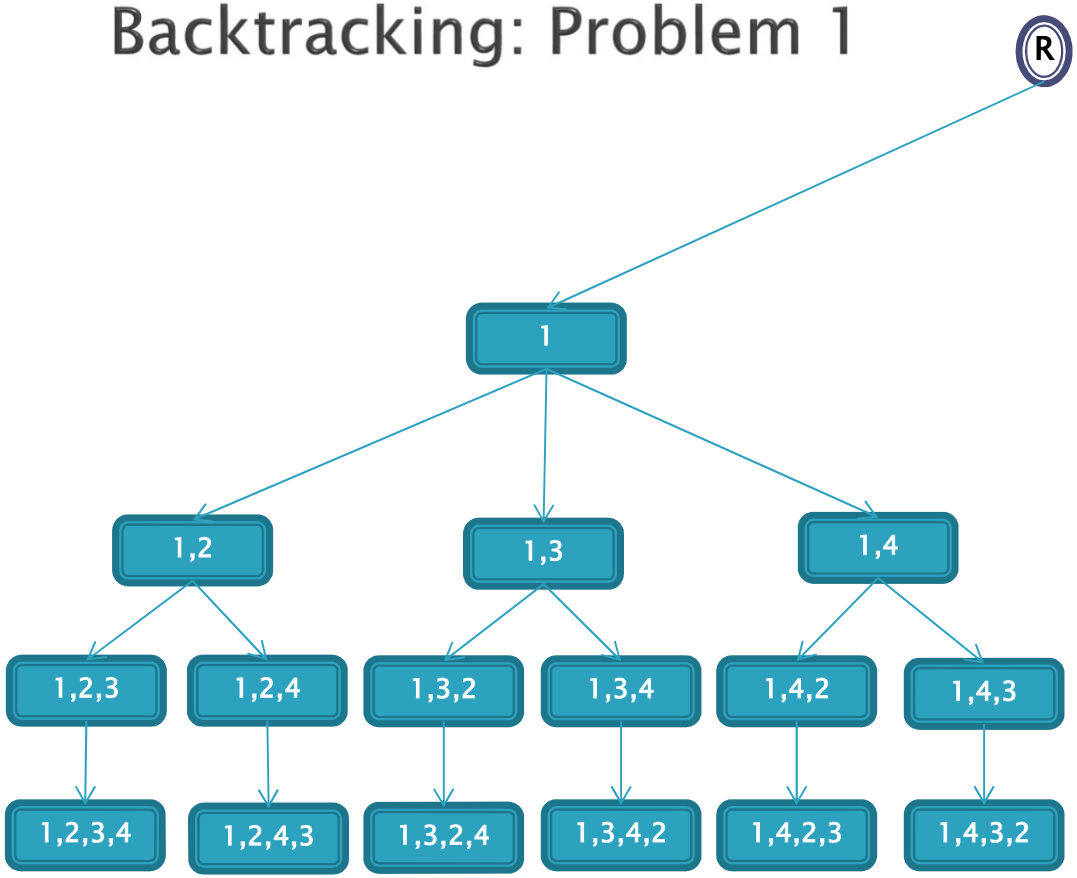
Backtracking: Problem 1

Input ={1,2,3,4}



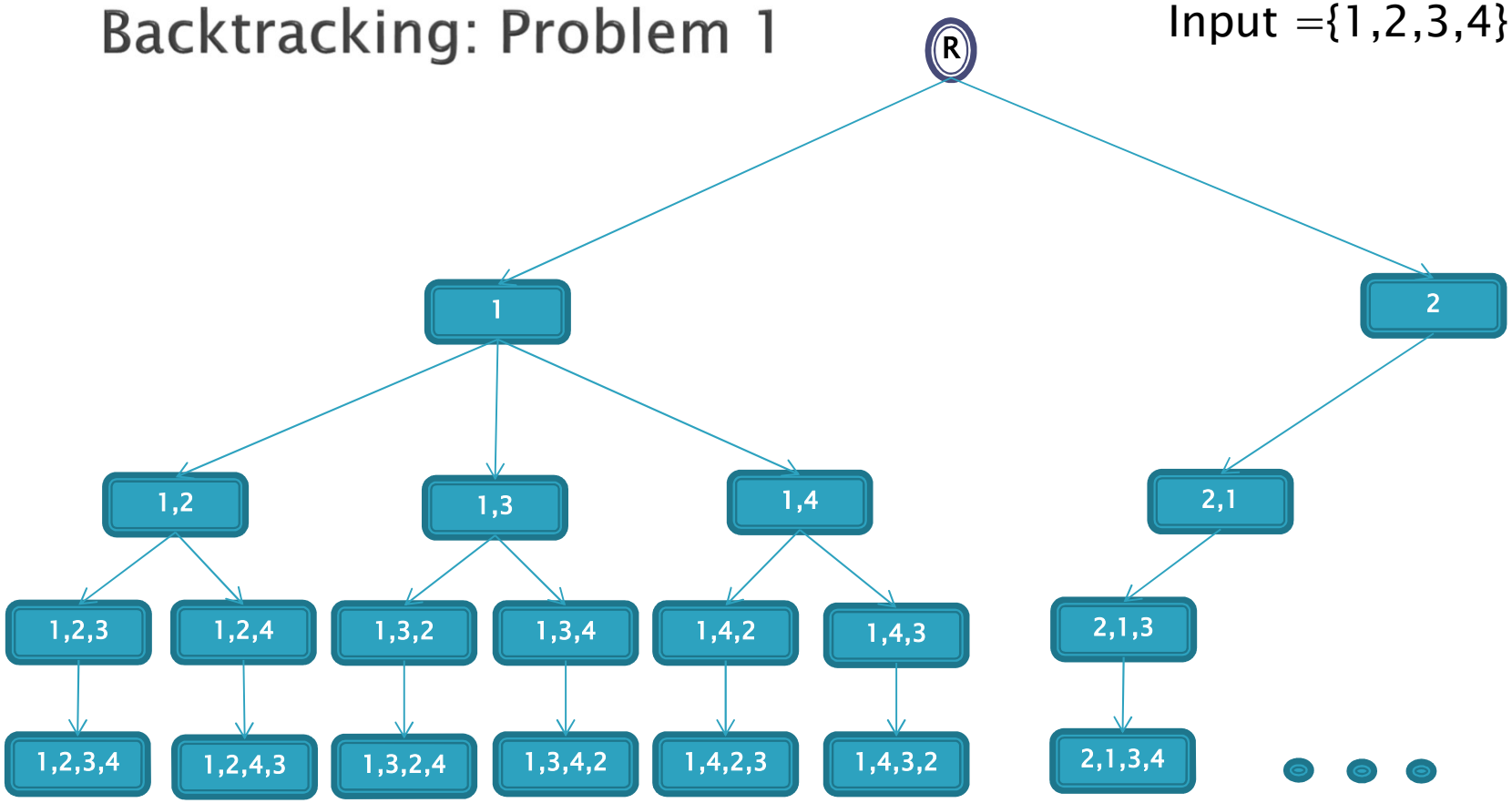
Backtracking: Problem 1

Input ={1,2,3,4}



Backtracking: Problem 1

Input ={1,2,3,4}

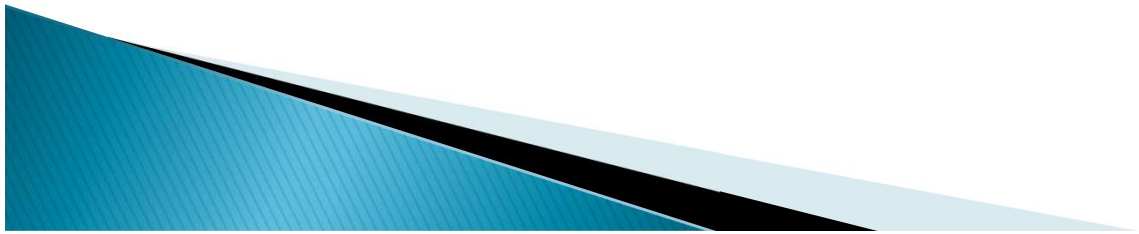


Backtracking: Problem 1

Input = [1, 2, 3, 4]									
Sol.	O[1]	O[2]	O[3]	O[4]	Sol.	O[1]	O[2]	O[3]	O[4]
1	1	2	3	4	13	3	1	2	4
2	1	2	4	3	14	3	1	4	2
3	1	3	2	4	15	3	2	1	4
4	1	3	4	2	16	3	2	4	1
5	1	4	2	3	17	3	4	1	2
6	1	4	3	2	18	3	4	2	1
7	2	1	3	4	19	4	1	2	3
8	2	1	4	3	20	4	1	3	2
9	2	3	1	4	21	4	2	1	3
10	2	3	4	1	22	4	2	3	1
11	2	4	1	3	23	4	3	1	2
12	2	4	3	1	24	4	3	2	1

Backtracking: Problem 1

- Inputs [1…N]: the vector of elements to interchange.
- Valids [1…N]: indicates if an element can be considered or not.
- Output [1…N]: valid output, progressively generated.
- All the calls in a brach start with the same conditions.
- The changes after each call must be undone.



Backtracking: Problem 1

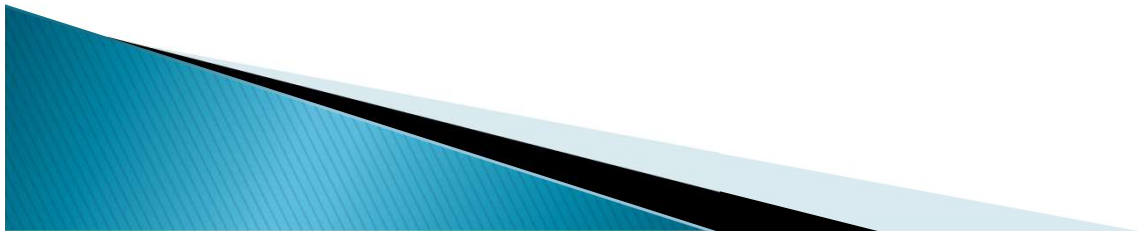
```
const N = ...
types bool = array[1... N] of boolean
types int = array[1... N] of integer


proc Initialize (I/O valids: bool)
   for i=1 to N do valids[i] = true efor
eproc

proc Problem1 (I/O input, output :int ; I/O valids :bool, I k:integer)
   var i : integer
   if k > N then
      Write output //If k (Depth) > Total Numbers the result is shown
   else
      for i=1 to N do
         if valids[i] then //If the number is a  candidate
            valids[i] = false //Candidate false
            output[k] = input[i]//The value is taken as partial result
            Problem1(input, output, valids, k+1) //Goes down in depth
            valids[i] = true //The data are restored in depth
         eif
      efor
   eif
Eproc
```

MAIN PROGRAM

Initialize (valids)
Problem1 (input, output, valids, 1)

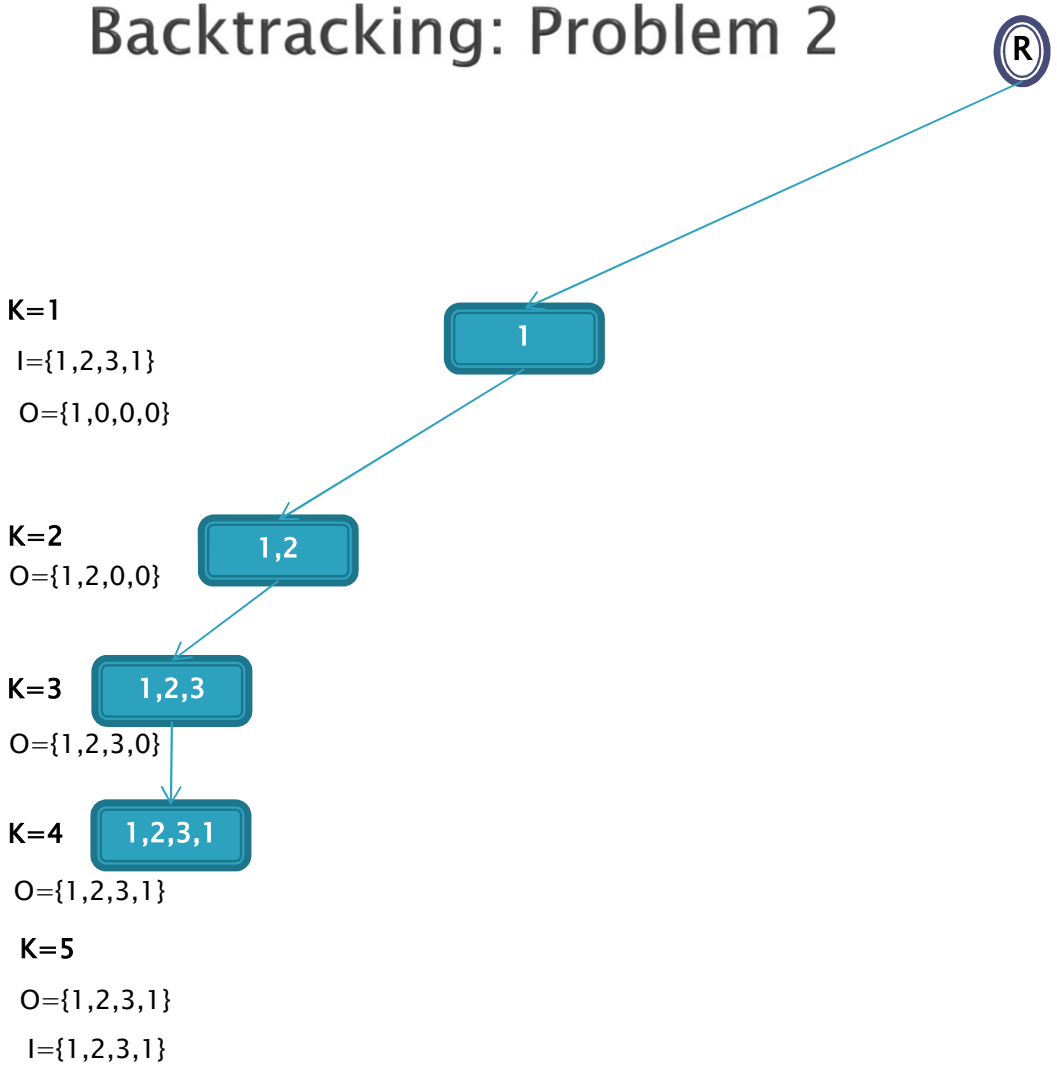
Backtracking: Problem 2

Solve the previous problem considering the possibility that the elements repeat themselves (for example, vector 1,2,3,1 or the chain acabada).



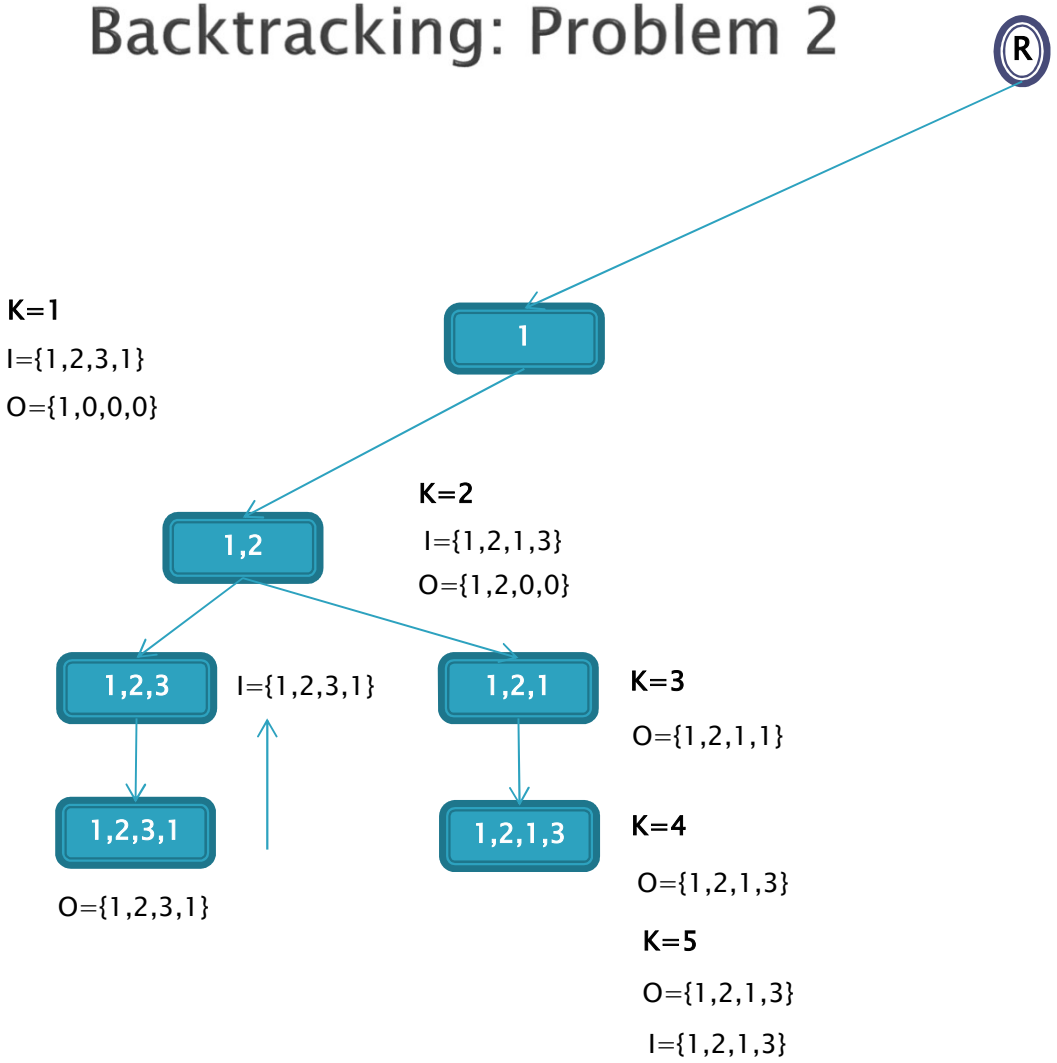
Backtracking: Problem 2

Input={1,2,3,1}



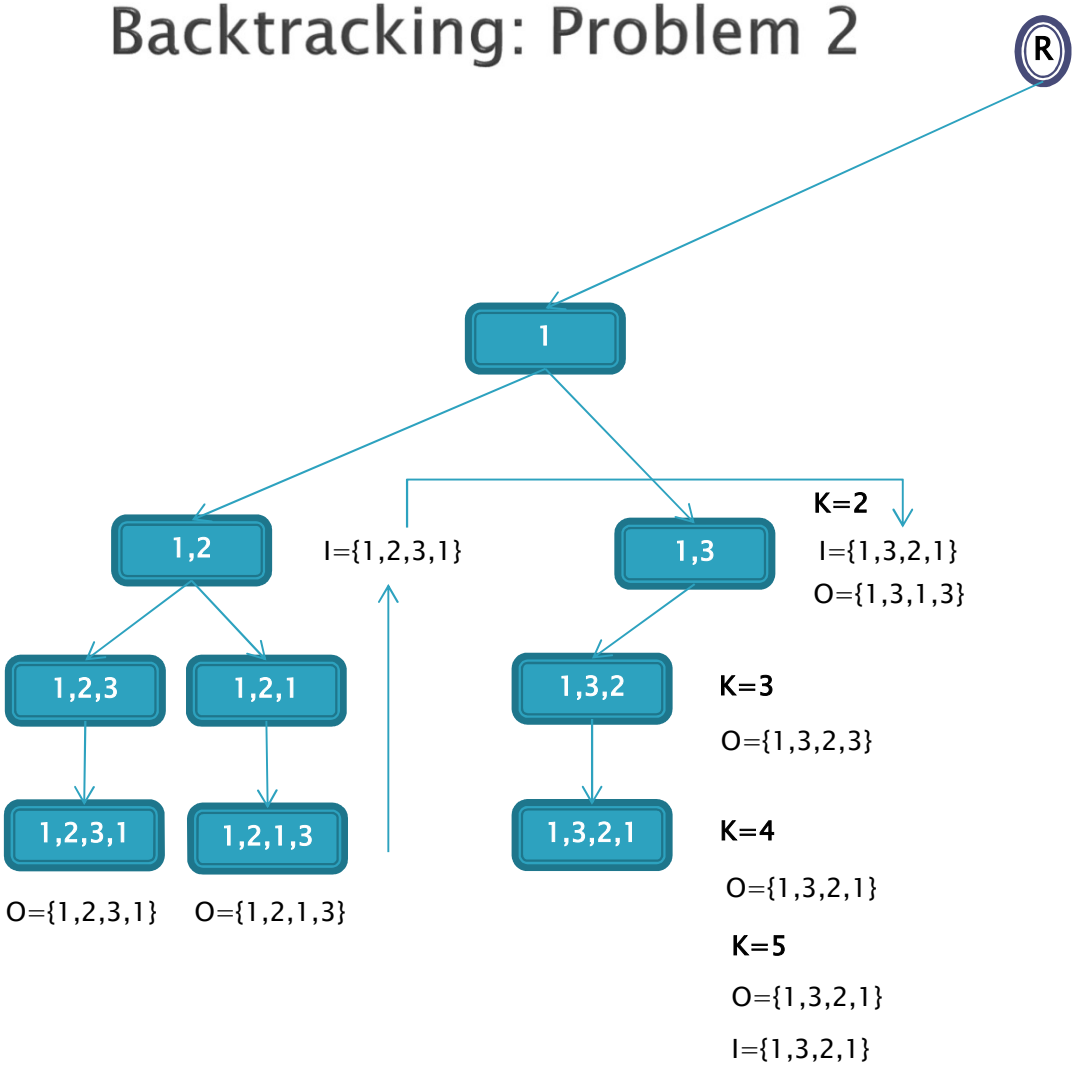
Backtracking: Problem 2

Input ={1,2,3,1}



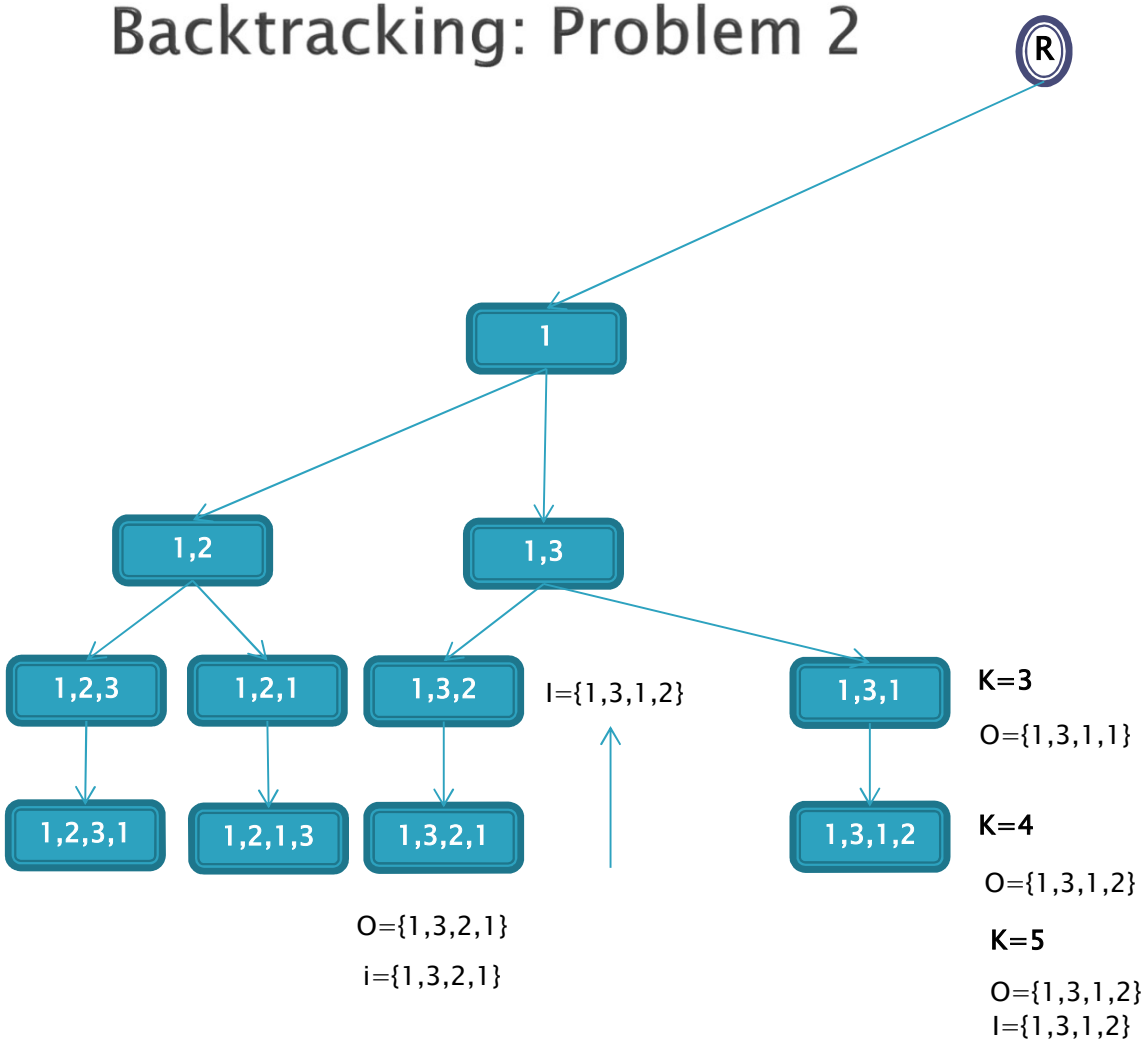
Backtracking: Problem 2

Input ={1,2,3,1}



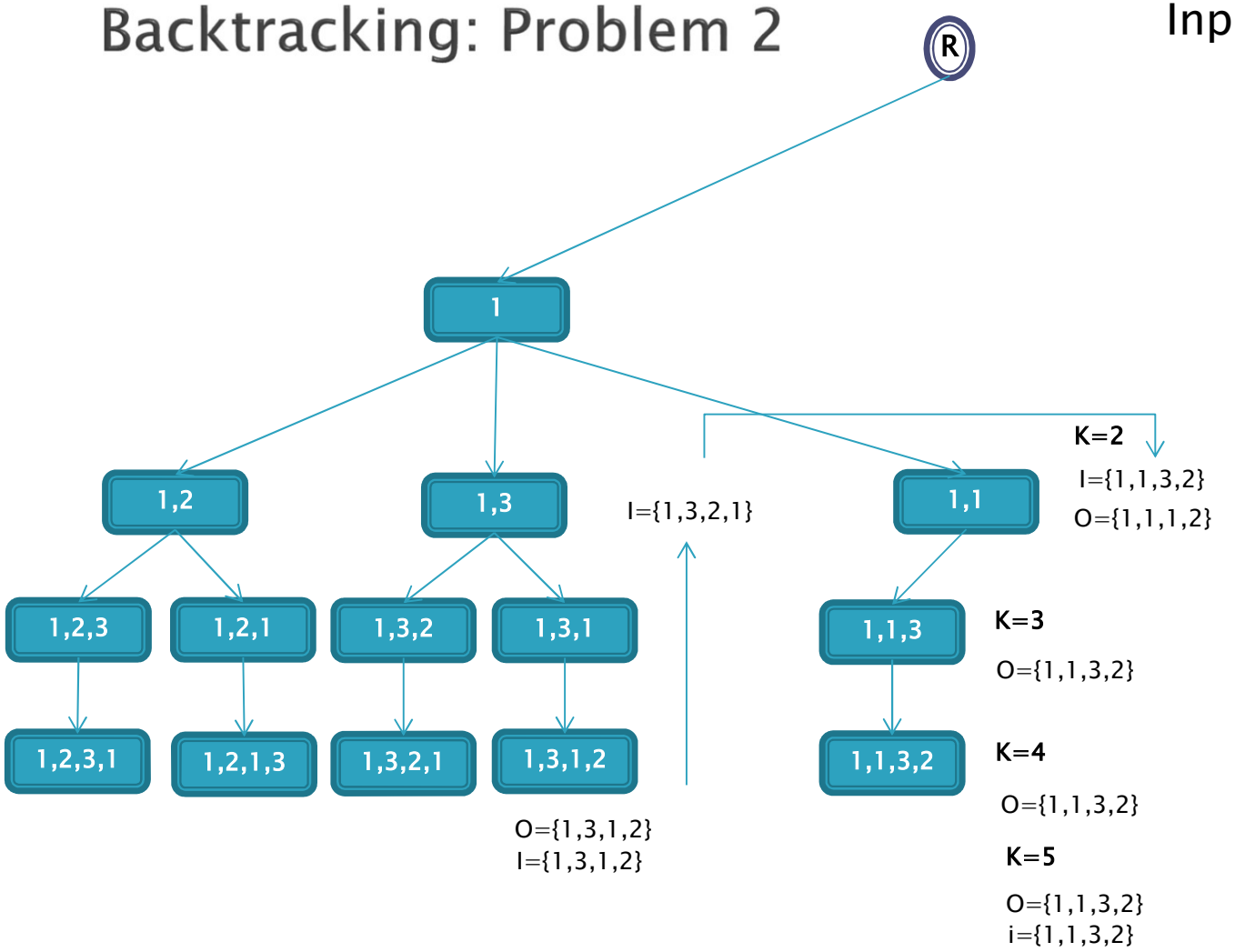
Backtracking: Problem 2

Input ={1,2,3,1}

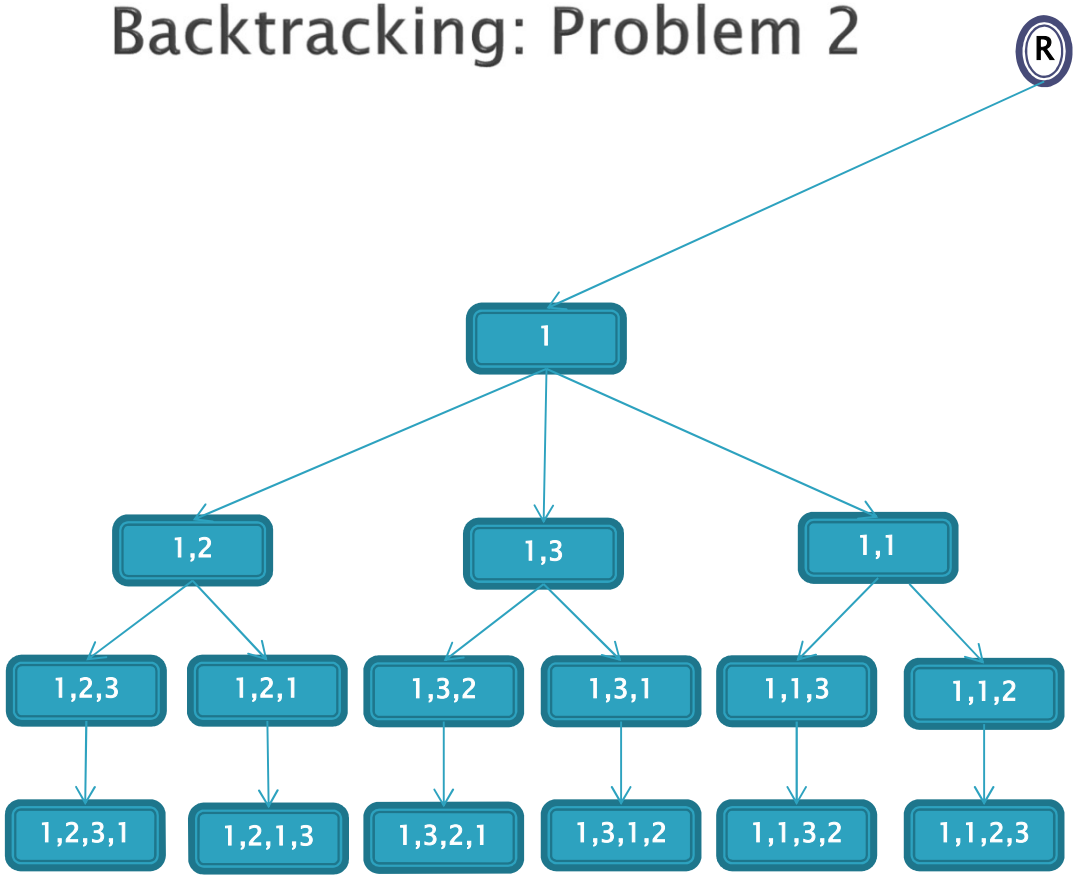


Backtracking: Problem 2

Input ={1,2,3,1}

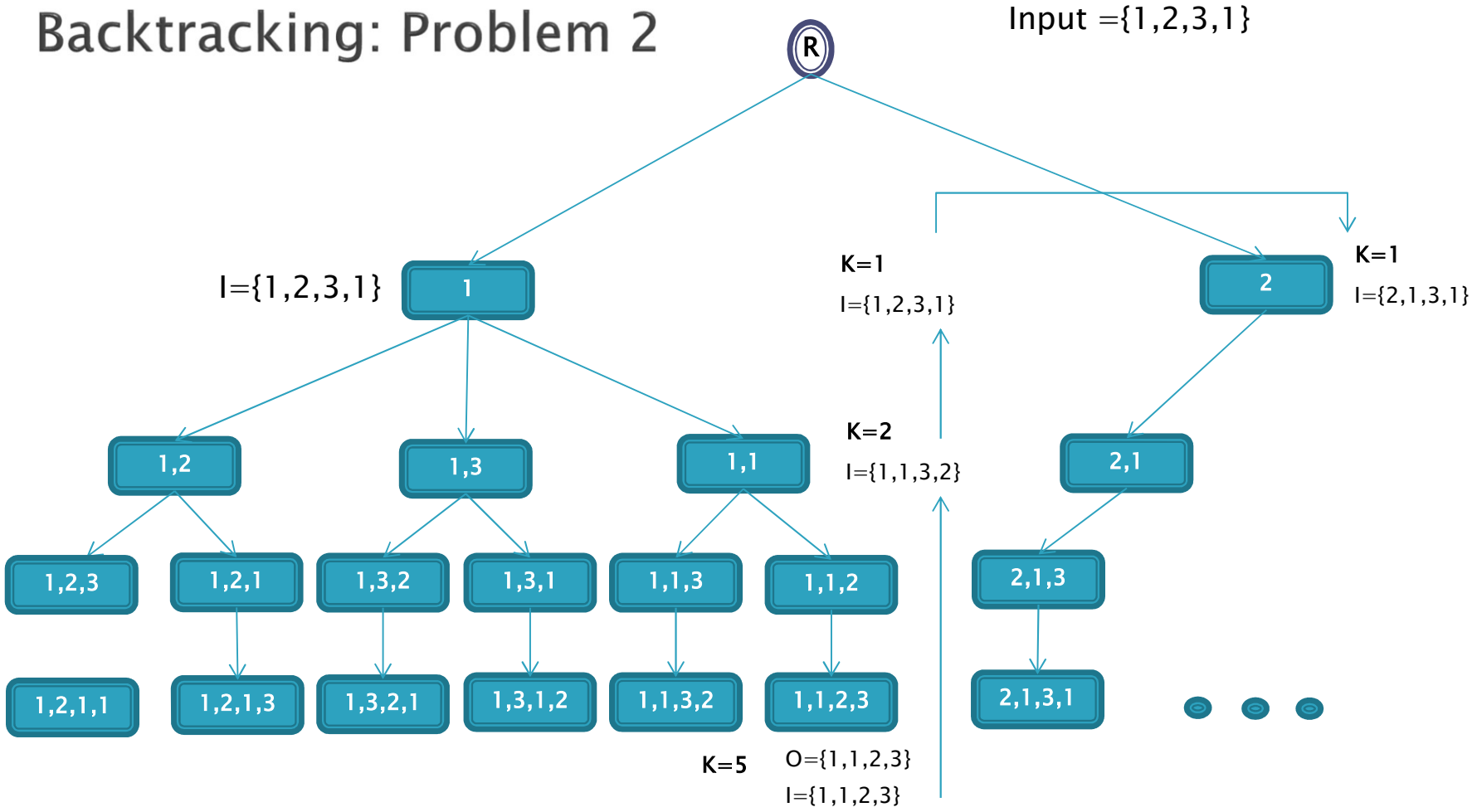


Backtracking: Problem 2



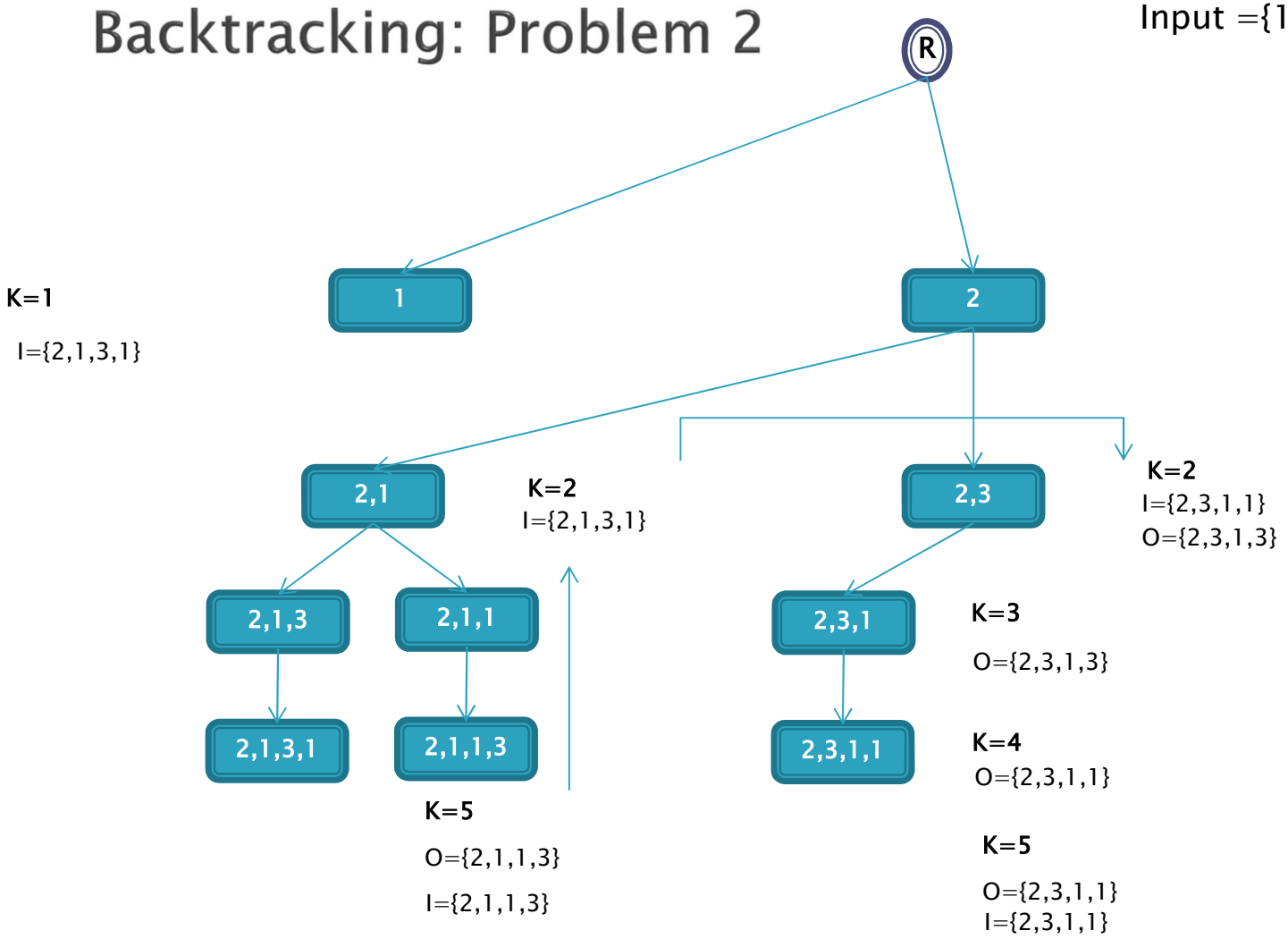
Input ={1,2,3,1}

Backtracking: Problem 2



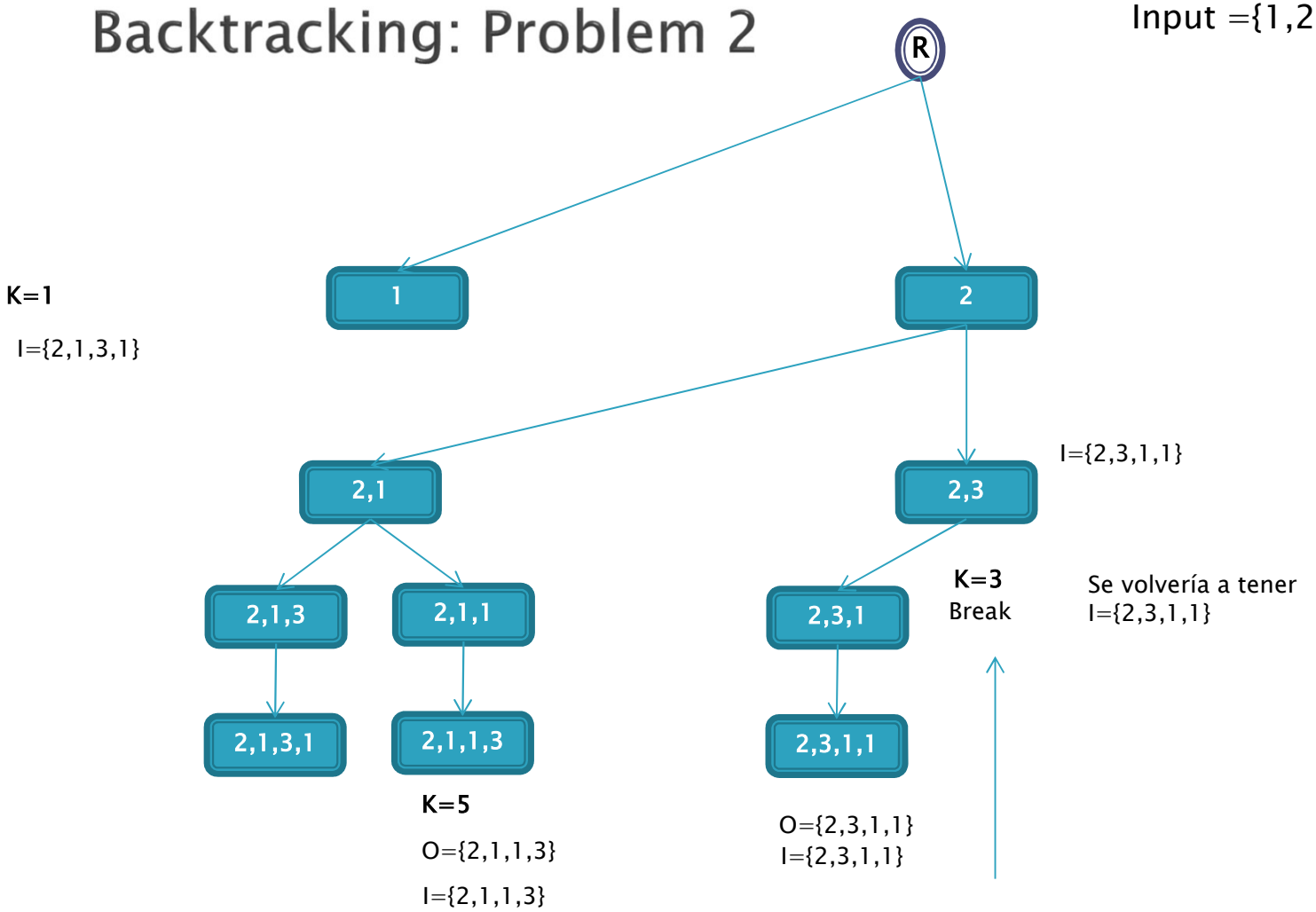
Backtracking: Problem 2

Input ={1,2,3,1}



Backtracking: Problem 2

Input ={1,2,3,1}



Backtracking: Problem 2

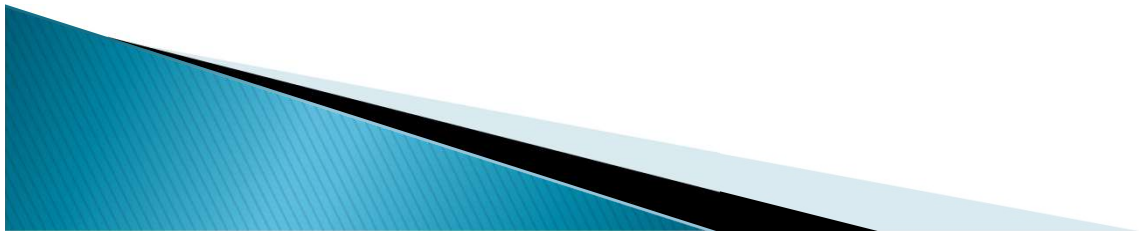
Input= [1, 2, 3, 1]									
Sol.	O[1]	O[2]	O[3]	O[4]	Sol.	O[1]	O[2]	O[3]	O[4]
1	1	2	3	1	13	3	1	2	1
2	1	2	1	3	14	3	1	1	2
3	1	3	2	1	~~15~~	~~3~~	~~2~~	~~1~~	~~1~~
4	1	3	1	2	16	3	2	1	1
6	1	1	2	3	~~17~~	~~3~~	~~1~~	~~1~~	~~2~~
5	1	1	3	2	~~18~~	~~3~~	~~1~~	~~2~~	~~1~~
7	2	1	3	1	~~19~~	~~1~~	~~1~~	~~2~~	~~3~~
8	2	1	1	3	~~20~~	~~1~~	~~1~~	~~3~~	~~2~~
9	2	3	1	1	~~21~~	~~1~~	~~2~~	~~1~~	~~3~~
~~10~~	~~2~~	~~3~~	~~1~~	~~1~~	~~22~~	~~1~~	~~2~~	~~3~~	~~1~~
~~11~~	~~2~~	~~1~~	~~1~~	~~3~~	~~23~~	~~1~~	~~3~~	~~1~~	~~2~~
~~12~~	~~2~~	~~1~~	~~3~~	~~1~~	~~24~~	~~1~~	~~3~~	~~2~~	~~1~~

Backtracking: Problem 2

Input = [1, 2, 3, 1]				
Sol.	O[1]	O[2]	O[3]	O[4]
1	1	2	3	1
2	1	2	1	3
3	1	3	2	1
4	1	3	1	2
5	1	1	3	2
6	1	1	2	3
7	2	1	3	1
8	2	1	1	3
9	2	3	1	1
10	3	2	1	1
11	3	1	2	1
12	3	1	1	2

Backtracking: Problem 2

- Inputs [1…N]: the vector of elements to interchange.
- Output [1…N]: valid output, progressively generated.
- All the calls in a brach start with the same conditions.
- The inputs are interchanged to avoid repeated solutions.
- The changes after each call must be undone.



Backtracking: Problem 2

```
const N = ...
types int = array[1... N] of integer

proc Interchange(I input:in, I i,j:integer)
   var a: integer
   a=input[i]; input[i]=input[j]; input[j]=input[i];
eproc

proc Problem2 (I input, output: I k: integer)
   var i, j, skip_iteration: integer
   if k>N then Write output //If k(Depth)>Total Numbers show result

   else
      for i=k to N do
         skip_iteration=0
         for j=k to i-1 do //If it is repeated I skip the iteration
             if  input[i]=input[j]then skip_iteration = 1; break for
             eif
         efor
         if  skip_iteration = 0 then
             output[k]=input[i] //It is taken as partial result
             Interchange(input,i,k) //The non selected are interchanged
             Problem2(input, output, k+1)
             Intercahange (input, k, i)//The original data are restored
         eif
      efor
   eif
Eproc
```

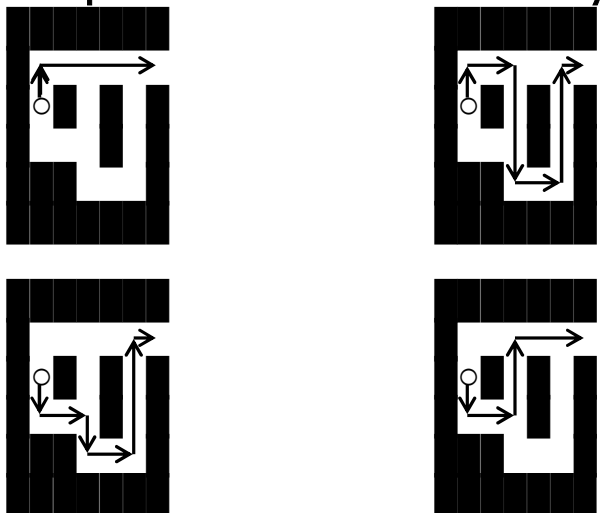
MAIN PROGRAM

Problem2 (input, output, 1)

Backtracking: Problem 5

There is a labyrinth table [1..n, 1..m] with logical values representing a labyrinth.The TRUE value indicates the existence of a wall (cannot be traversed), while FALSE represents a recordable box.

To move through the labyrinth, you can move horizontally or vertically from one square, but only to an empty square (FALSE). The edges of the table are completely TRUE except one box, which is the output of the labyrinth. Design a Backtracking algorithm that finds all the possible paths that lead to the exit from a certain initial box, if it is possible to leave the labyrinth. Design a Backtracking algorithm that finds the best possible path that leads to the exit from a certain initial box, if it is possible to leave the labyrinth.



Backtracking: Problem 5

Strategy:

1. Check if the box where it is currently is the output, if so, leave the function indicating that the solution has been found; otherwise go to the next step.

2. If possible, visit the left boxes by recursively calling the same function. Verify step 1

3. If it is not possible to visit or find the exit by step 2, look for the exit with the box above. Verify step 1

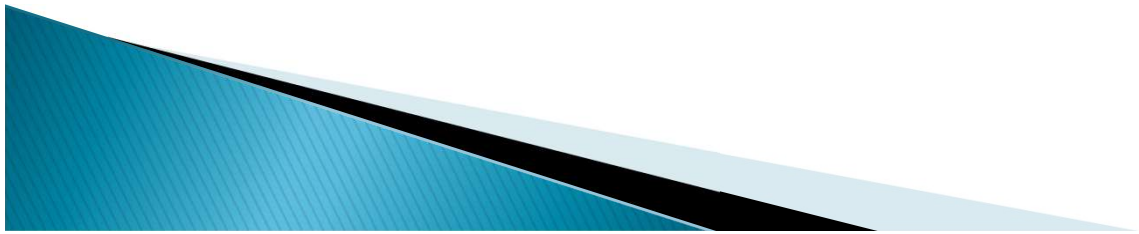
4. If it is not possible to visit or find the exit by step 3, look for an exit with the box on the right. Verify step 1

5. If it is not possible to visit or find the exit by step 4, look for an exit with the box below. Verify step 1

6. If the exit was not found, exit the function indicating that there is no exit through the current square.

Backtracking: Problem 5

- Labyrinth[R][C]. Matrix which stores the labyrinth's structure.
  - "#" → Wall of the labyrinth → This cell cannot be reached.
  - "O" → Starting point.
  - "S" → Exit.
  - "." → Path



Backtracking: Problem 5

```
Proc travel(I labyrinth[R][C]:Matrix of characters, I i:integer, I j:integer){
      //It is checked if it is a solution
      if(labyrinth[i][j]=='S')then
            print(labyrinth);
            labyrinth[i][j]='Y';
            existSolution=true;
            return;
      if
      labyrinth[i][j]='.';//The path is marked

      //left or exit
      if(i-1>=0 && i-1<F &&(laberinto[i-1][j]==' '||laberinto[i-1][j]=='S'))then
            travel(labyrinth, i-1, j);
      //up or exit
      if(j+1>=0 && j+1<C && (laberinto[i][j+1]==' ' || laberinto[i][j+1]=='S'))then
            travel(labyrinth, i, j+1);
      //right or exit
      si(i+1>=0 && i+1<F && (laberinto[i+1][j]==' ' || laberinto[i+1][j]=='S'))then
            travel(labyrinth, i+1, j);
      //down or exits
      if(j-1>=0 && j-1<C && (laberinto[i][j-1]==' ' || laberinto[i][j-1]=='S'))then
            travel(labyrinth, i, j-1);
      labyrinth[i][j]=' ';//Se desmarca el camino
eProc
```

MAIN PROGRAM

```
travel(labyrinth, start_x, start_y);
```

Backtracking: Problem 5

Solution